REMARKS

Claims 2, 4-29, 31-50, 52-64, 66-78, and 162-169 are pending in the present application. Claims 2, 27, 48, 64, 78, 162, 163, 164, 165, and 166 are independent claims.

All pending claims stand rejected under 35 U.S.C. § 103(a).

Reconsideration is respectfully requested in view of the following remarks.

Rejections Under 35 U.S.C. § 103

All pending claims stand rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over U.S. patent number 5,987,247 (hereinafter "Lau") in view of U.S. patent number 6,337,696 (hereinafter "Lindhorst") and U.S. patent number 6,023,271 (hereinafter "Quaeler-Bock").

Applicant respectfully submits that the Office cannot meet its legal burden to establish a prima facie case of obviousness of the amended claims under 35 U.S.C. § 103(a) based upon the cited references. Reconsideration is respectfully requested.

Applicant discloses methods and systems for generating a web application. In an exemplary method, a web application server receives input files from graphic designers or business analysts. The input files may comprise at least one web application graphical user interface. The web application server determines if an application framework code is available for the web application, and retrieves the application framework code from an application directory. If the application framework code is not available for the web application, then the web application server generates the application framework code, along with a business logic foundation code, an event handler skeleton and a graphical user interface code. Generating the event handler skeleton comprises parsing the input files, reviewing each parsed input file for a tag type, attribute name and attribute value, and determining an event handler method based on the tag type, attribute name and attribute value.

In particular, in connection with generating the event handler skeleton, the application specification explains as follows:

> After identifying the input tag, web application generator 205 searches for specific attributes within the input tag, based on the input tag type. In one embodiment, certain tags have a single attribute, while other tags have multiple attributes. After identifying each attribute within the tag, web application generator 205 identifies an attribute value associated with the attribute name. *Ultimately,* ***based on*** *the input tag, attribute name(s) and associated attribute value(s), web application generator 205* ***relies on a particular "rule" (or fixed formula) within web application server memory 325 to generate event handler code*** *620 and GUI code 625 (step 730). Thus, the input tag, attribute name and attribute value determine the "rule" that web application generator 205 relies on to generate event handler code 620 and GUI code 625.*
>
> *For example, in one embodiment, one tag of an Input file in HTML format is "<input type=submit name=JLMC\|ick value=Add>". Web application generator 205 first identifies the tag as an "input" tag, next identifies the attribute names as "name" and "value", and further identifies the associated attribute values as "JMLClick" and "Add".* ***Based on the input tag, attribute names and associated attribute values, web application generator 205 relies on a particular rule in web application server memory 325 to generate the following event handler method****: "public void add (App app) throws Exception", as well as a GUI code in the form of a source file such as "indexClick.java." If one of the attribute values was something other than JLMClick, then web application generator 205 would have relied on a separate rule in web application server memory 325 to generate event handler code 620 and GUI code 625.* (Application Specification, p. 15)

The web application server receives web application business logic objects and event handlers from the web developers, and organizes the application framework code, web application business logic objects and event handler methods into web application source code. The web application server then compiles the web application source code. Modified input files may subsequently be received by the web application server from the graphic designers or business analysts. The modified input files are compiled and dynamically bound with the compiled web application source code at runtime.

Claim 1 recites as follows:

> A computer-implemented method of generating computer code for a web application, comprising:
> a computing system receiving an input file, the input file comprising markup language text for a graphical user interface;
> the computing system generating an event handler skeleton, wherein generating an event handler skeleton comprises:
> **the computing system parsing the markup language text;**
> **the computing system identifying an input tag in the markup language text;**
> **the computing system identifying in the markup language text, an attribute within the markup language tag;**
> **the computing system identifying in the markup language text a value corresponding to the identified attribute;**
> **the computing system using the identified markup language input tag, the identified attribute, and the identified value to identify a rule for creating an event handler method; and**
> **the computing system using the identified rule and the identified markup language input tag, the identified attribute, and the identified value to create an event handler method;**
> the computing system receiving web application business logic objects;
> the computing system receiving event handler methods;
> the computing system organizing the application framework code, the web application business logic objects and the event handler methods into application source code; and
> the computing system binding the web application source code with the input files at runtime.

In order for a set of references to render claim 1 obvious, the references must disclose or suggest the entirety of the recited method, including the **emphasized** language. The undersigned respectfully submits that cited references do not disclose at least the above-emphasized claim language and therefore cannot possibly disclose the recited combination.

The Office has acknowledged that ***Lau*** does not teach the previously recited language of "generating an event handler skeleton compris[ing] . . . parsing at least one input file; reviewing the parsed input file for one or more of a tag type, an attribute name and an attribute value; and determining an event handler method based on one or more of the tag type, the attribute name and the attribute value." (Final Office Action dated may 17, 2006, p. 3). Instead, the Office relies on Lindhorst as relevant to this claim language.

In the method disclosed in ***Lindhorst***, an HTML file is parsed into objects, HTML text, and scripts. (Col. 11 at ll. 41-44). The parsed objects are then listed in an event pane of a user interface and actions are listed in an action pane of the user interface. (Col. 11 at ll. 53-55). The user matches an action with an event using the user interface. (Col. 11 at ln. 59 – Col. 12 at ln. 10). Based upon the user's selections, a script is generated to match the event with an action. (Col. 12, ll. 13-17).

Accordingly, Lindhorst discloses a process wherein user inputs are employed to match HTML interface events with existing methods, and then scripts are generated from the methods. But in contrast with the amended language of claim 2, Lindhorst does not disclose "[a] computing system using the identified markup language, tag, the identified attribute, and the identified value **to identify a rule for creating an event handler method**." Indeed, there is no discussion at all in Lindhorst of a "**rule** for **creating** an event handler," and there certainly is no discussion of "using the identified markup language, tag, the identified attribute, and the identified value **to identify the rule**." Furthermore, Lindhorst does not disclose "the computing system **using the identified rule** and the identified markup language input tag, the identified attribute, and the identified value **to create an event handler method**."

The Board of Appeals referenced a section of Lindhorst that discloses creating a script from an existing method. (See Lindhorst at col. 20, ll. 10-25). Thus, in this referenced section of Lindhorst, the method ***already exists***, whereas in the recited claim language "an even handler method" ***is being created***. Further, in the portion of Lindhorst referenced by the Board of Patent Appeals, Lindhorst discloses generating an invocation string "call Button.Show()" from the "Show" method. Lindhorst further discloses that if there had been parameters on the method,

the parameters would have been incorporated into the script, e.g., "call Button.Show(x, y)." But in contrast with the recited claim language, translating a method into a script as disclosed by Lindhorst is not the same or similar "using the identified markup language, tag, the identified attribute, and the identified value **to identify a rule for creating an event handler method.**" Likewise, translating a method into a script as disclosed by Lindhorst is not the same or similar to "**using the identified rule** and the identified markup language input tag, the identified attribute, and the identified value **to create an event handler method**."

The remaining reference relied upon for the rejection, ***Quaeler-Bock***, discloses a system for programming graphical user interfaces. (Abstract). The Office cites to Quaeler-Bock as being relevant to binding components to objects. The Office does not allege that Qualer-Bock is relevant to "generating an event handler." And in fact, Quaeler-Bock does not teach or suggest "[a] computing system using the identified markup language, tag, the identified attribute, and the identified value **to identify a rule for creating an event handler method**." Furthermore, Lindhorst does not disclose "the computing system **using the identified rule** and the identified markup language input tag, the identified attribute, and the identified value **to create an event handler method**."

Therefore, because the cited references do not teach or suggest the emphasized language, even in combination the references do not disclose or suggest the combination recited in claim 2. Accordingly, the Office cannot establish a prima facie case of obviousness of the amended claims under 35 U.S.C. § 103(a) based upon the cited references. Independent claims 27, 48, 64, 78, 162, 163, 164, 165, and 166, while their claim language is different from that of claim 1, are novel and non-obvious for reasons similar to those discussed above.

Reconsideration and withdrawal of the rejections under 35 U.S.C. § 103 is respectfully requested.

CONCLUSION

The undersigned respectfully submits that pending claims are allowable and the application in condition for allowance. A Notice of Allowance is respectfully solicited.

Examiner Wood is invited to call the undersigned in the event a telephone conference would advance prosecution of this application.

Date: June 30, 2010

/John E. McGlynn/
John E. McGlynn
Registration No. 42,863

Woodcock Washburn LLP
Cira Centre
2929 Arch Street, 12th Floor
Philadelphia, PA 19104-2891
Telephone: (215) 568-3100
Facsimile: (215) 568-3439